INFO6205 Final Project
Genetic Algorithms

Rui Xia 001417136 (Sec 03)
Yumeng Xiao 001443049 (Sec 05)
Yixuan Wang 001494410 (Sec 05)

Summary

We wanted to implement GAs to simulate image generation, so we chose the target graph just with 2 colors: white and black. At first, the program will read the image and will produce the corresponding Boolean array to store the information of RGB/ color point in this image. And then it will initiate **every row *(individual)*** array in the first generation randomly, push them into the simulation of the Genetic Algorithm, randomly change the **RGB/color point *(gene)*** of individual by natural Mutation and Crossover based on different probabilities, select each *set of rows* ***(generation)*** in the process of evolution based on the ***fitness function***. Eventually, we will tend to select the ideally best row and make them form the best graph.

The target graph as following:

File Description

INFO6205_Project_src_517: The source file of Java Program
original. jpg: The original graph used in the program

If you run the program, there are extra 5 picture file will be generated in the folder.

Structure and Details

- Problem

Our target is an 80 x 80 picture. We aim to generate picture whose pixels have the correct color and use the genetic algorithm to produce the target picture.

- Genotype

For each gene which represents the color of the pixel, it only has two value: true and false, representing white and black.

- Fitness

We calculate fitness by comparing each pixel's color or which be represented by each gene's value to origin image's. After getting the number of the gene with the right value, we used it to divide the total number of origin image to a form of a percentage.

```
public double getFitness(boolean[] ind, boolean[] originalImage, int len) {
    double fitness = 0;
    for(int i = 0; i < len; i++) {
        if(originalImage[i] == ind[i])
            fitness++;
    }
    return fitness/len;
}
```

- Mutation

Totally Random Mutation

```
private void mutation(boolean[] individual) {
    int length = individual.length;
    Random r = new Random(System.currentTimeMillis());
    individual[r.nextInt(length)] ^= true;//Change black to white/white to black
}
```

Partially Directed Mutation

```
private void mutation_Xmen(boolean[] individual){
    int len = originalImage.length;
    boolean[] Xmen = new boolean[len];
    System.arraycopy(originalImage, 0, Xmen, 0, len);
    // Change 10% color point of original image
    int count = (int) (len * (0.1));
    for(int i = 0; i < count; i ++) {
        Random r = new Random(System.currentTimeMillis());
        int n = r.nextInt(len);
        Xmen[n] ^= true;
    }
    // Make new individual have the 90% fitness value
    System.arraycopy(Xmen, 0, individual, 0, len);
}
```

- Crossover

```
private void cross(boolean[] arr1, boolean[] arr2) {
    Random r = new Random(System.currentTimeMillis());
    int length = arr1.length;
    int slice = 0;
    do {
        slice = r.nextInt(length);
    } while (slice == 0);
    if (slice < length / 2) {
        for (int i = 0; i < slice; i++) {
            boolean tmp = arr1[i];
            arr1[i] = arr2[i];
            arr2[i] = tmp;
        }
    } else {
        for (int i = slice; i < length; i++) {
            boolean tmp = arr1[i];
            arr1[i] = arr2[i];
            arr2[i] = tmp;
        }
    }
}
```

Process

- Firstly, we used a pixel-number length boolean array as the expression of an individual. Initialed all pixel with random TRUE(black) or FALSE(white) and used crossing and mutation to involve. However after thousands, we found the max fitness didn't increase even decline at first, and finally fluctuate up and down around 60 percent.

- Then, We thought it might because the differences between offspring and father generation are not big enough. So actually all the offsprings generated after would all be similar to the first generation. And with the elimination in each generation, the offspring will become just similar to only a few individuals whose fitness is higher than others in the first generation but hard to generate new individuals who can exceed the first generation.
  So we write a new function to implement the gene's recombination when parents mate to get a new individual of the next generation.

```
public List<boolean[]> mating(double[] fit, List<boolean[]> parent ) {
    List<boolean[]> newInd = new ArrayList<boolean[]>(HEIGHT);

    List<boolean[]> parent1 = parent;
    List<boolean[]> parent2 = individuals.get(nextDiscrete(fit));

    for(int i = 0;i<chrom_len;i++) {
        boolean[] chrom1 = parent1.get(i);
        boolean[] chrom2 = parent2.get(i);

        Random r = new Random();

        if(r.nextDouble() < cross_ratio)
            cross(chrom1,chrom2);

        if (r.nextDouble() < muta_ratio) {
            mutation(chrom1);
        }

        int fit1 = 0;
        int fit2 = 0;

        for(int j = 0;j<gene_len;j++) {

            if(chrom1[j]==originalImage[i*WIDTH+j])
                fit1++;
            if(chrom2[j]==originalImage[i*WIDTH+j])
                fit2++;
        }

    int num = r.nextInt(2);
        if(num==0) newInd.add(chrom1);
        else newInd.add(chrom2);
    }

    return newInd;


}
```

  But fitness still hasn't changed and improve too much.

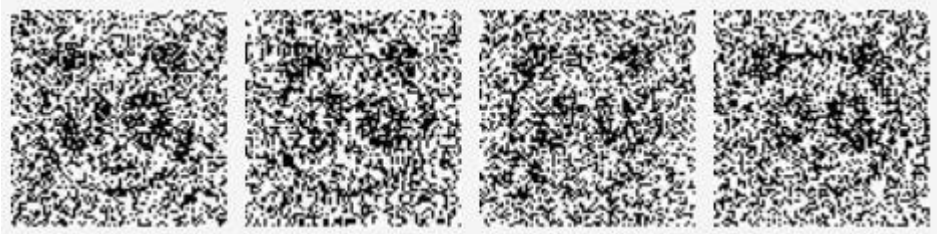
- After that, we thought about getting the first generation with enough different fitness. Because in this way, we can provide more possible situations for offspring to inherit and evolve, which might be helpful to get better individuals. So we initiated our first generations with evenly distributed fitness from 0 to 1.

```
public void initPopulation() {

    for (int i = 0; i < population; i++) {
        List<boolean[]> ind = new ArrayList<boolean[]>(HEIGHT);
        for(int j = 0;j<chrom_len;j++) {
            boolean[] chrom = new boolean[gene_len];
            System.arraycopy(originalImage, j*WIDTH, chrom, 0, gene_len);
            ind.add(chrom);
        }

        Random r = new Random();
        int amount = r.nextInt(6400);
        for (int j = 0; j < amount; j++) {
            int line = r.nextInt(chrom_len);
            boolean[] chrom = ind.get(line);

            int dot= r.nextInt(gene_len);
            boolean b = chrom[dot];
            chrom[dot] = !b;
        }
        individuals.add(ind);
    }

}
```

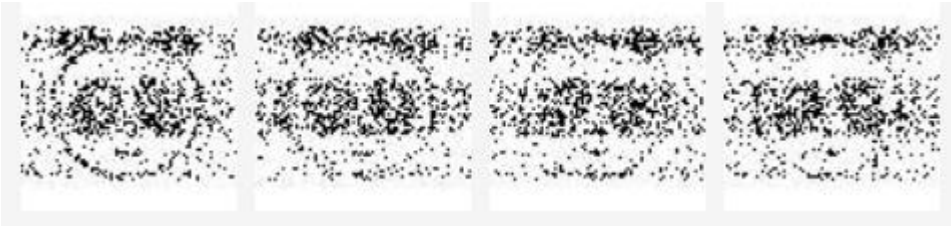
At this time, we get the result of continued fitness and even reached more than 80 percent. But the output of the generated image showed that the image white at last. We think it is because our target image has much more white pixel than black so all the offspring finally tended to be white.

test.jpg

- After the first totally failed version, we changed our individual to each line of the target photo. Randomly initialed every pixel in each line. Each line evolved separately according to the different proportion of black and white pixel and then put them together. It could show an approximate contour now but still had no significant progress of fitness.



- Then we used a kind of rigged method -- read each line and recorded the number of black pixels. Initialed the same number of black pixels, which meant the only difference between individuals and the target photo was the positions of the black pixels on the target photo. It worked remarkably on white lines.



However, obviously, evolution progress was still little.

- It is really worth to mention that we find the choke point in the process -- our evolution even started to degenerate after a certain number of generation. We tried to change the mutation rate, the cross rate, the initialization function and value. We found that excessive mutation ratio will destroy the gene of excellent individuals and let the great gene can't be inherited. Also, we found out population could control the evolution. The population was the number of individuals in each generation. We kept it as 250 in previous experiments, while we had about 6400 pixels in the target picture. We changed population to width times height of the target photo and got a better evolution. We can notice this phenomenon in the following pictures, they are different generated graphs based on the different limits of the number of generation (100, 1000, 5000, 10000).

100 1000 5000 10000

- So we want to explore the reason of degeneration and try to deal with this problem. In our case, we found that the number of white points in this target graph is greatly more than the number of black points. So maybe mutation is more likely to choose white points and change them to black. It is absolutely not good for our evolution. So we decide to make an interesting experiment: we set a really small probability of mutation, almost half of the probability of original random mutation, as directed mutation, which means that points which are chosen by this mutation will be changed to be "Supermen" / "X-Men" with the really high fitness value, about 0.9. As you can see in the following graphs, this strategy works very well. When the limit of the number of generation equals 10000, the generated graph is almost the same as the original/ target one.

100 1000 5000 10000

Conclusion

- Crossover ratio and mutation ratio should be suitable or they will destroy the gene of good individuals and the hinder the evolution.
- The population should be big enough to get more good new individuals and prevent degeneration.
- For finding the best result, we can try to keep the differences between individuals and each generation be big enough. So it can help us to find the optimal solution.
- Fitness value may float around a certain value, the evolution will make them tend to just find the best solution in that small solution spaces/ local optimal solution rather than the global optimal solution. For dealing with this problem, we can let the first generation cover more kinds of situation of array at the first place, or we can design gene combination to increase the diversity of offspring. Moreover, we also can design our mutation to be stronger to get more new and good individuals for the next generation, like X-men directed mutation method mentioned before.

Screenshot

Unit test